Drone Navigation and Avoidance of Obstacles Through Deep Reinforcement Learning

Ender Çetin, Cristina Barrado, Guillem Muñoz, Enric Pastor
Escola dEnginyeria en Telecomunicacions i Aerospacial de Castelldefels
Universitat Politecnica de Catalunya , BarcelonaTECH
Castelldefels, Spain
ender.cetin@upc.edu

***Abstract*—Unmanned aerial vehicles (UAV) specifically drones have been used for surveillance, shipping and delivery, wildlife monitoring, disaster management etc. The increase on the number of drones in the airspace worldwide will lead necessarily to full autonomous drones. Given the expected huge number of drones, if they were operated by human pilots, the possibility to collide with each other could be too high.**

In this paper, deep reinforcement learning (DRL) architecture is proposed to make drones behave autonomously inside a suburb neighborhood environment. The environment in the simulator has plenty of obstacles such as trees, cables, parked cars and houses. In addition, there are also another drones, acting as moving obstacles, inside the environment while the other drone has a goal to achieve. In this way the drone can be trained to detect stationary and moving obstacles inside the neighborhood and so the drones can be used safely in a public area in the future. The drone has a front camera and it can capture continuously depth images. Every depth image, with a size of 144x256 pixels, is part of the state used in DRL architecture. Also, another part of the state is the distance to the geo-fence, a virtual barrier on the environment, which is added as a scalar value. The agent will be rewarded negatively when it tries to overpass the geo-fence limits. In addition, angle to goal and elevation angle between the goal and the drone will be used as information to be added to the state. It is considered that these scalar values will improve the DRL performance and also the reward obtained. The drone is trained using Q-Network and its convergence and final reward are evaluated. The states containing image and several scalars are processed by neural network that joints the two state parts into a unique flow. This neural network is named as Joint Neural Network (JNN) [1]. The training and test results show that the agent can successfully avoid any obstacles in the environment. In training, there exist some episodes crashed at the beginning of the training session because the random drones moves randomly in the environment and thus they can hit the learner drone during training. The test results are very promising and the learner drone reaches the destination with a success rate %100 in first two tests and with a success rate %98 in the last test.

***Index Terms*—Drones, UAV, Deep Reinforcement Learning, Q-Network, DDQN, JNN**

I. Introduction

The usage of artificial intelligence (AI) in unmanned system technologies increases drastically. The main subjects where the AI are used include design of control systems and design an automatic flight control systems for UAVs. These control systems are used to control, navigate the drones and also avoid any kind of obstacles. Considering navigation of UAV, classical control methods which are used for controlling and navigating of the drones are not enough to detect and avoid obstacles during navigating inside the environment. In other words, the users can control the drones and navigate easily by selecting way-points inside the environment. However, navigating by following only specific positions without knowing whether there is an obstacle on the trajectory followed or not can cause collisions and then most of the cases there can be a full crash of an UAV.

In a study about UAV autonomous navigation system for GNSS (Global Navigation Satellite System) invalidation [2], it is stated that the GNSS which is used to correct the result of inertial navigation and also improve the navigation system can help navigating an UAV in most of the application scenarios. However, when the GNSS is not available, for example passing through the sheltered areas, it is necessary to use other methods to aid inertial navigation. To overcome this problem, the researchers used proper visual sensors for navigation. In another research [3], reinforcement learning algorithm is used to navigate an UAV in an unknown environments. In this study it is shown that the quadrotor can successfully learn how to navigate through an unknown environment by using Q-learning methods combined PID (Proportional-Integral-Derivative) controller.

In this paper, deep reinforcement learning algorithm is proposed to make drones behave autonomously inside a suburb neighborhood environment. Training of the drone can take almost up to 48 hours to finish and tests include 100 full autonomous episodes in unknown simulated environment.

The remaining of the paper is organized as follows. In section II, the theory about the reinforcement learning, neural networks and deep reinforcement learning is provided. Section III provides methods and supplementary information about the tools and DRL model including the environment, states, actions and rewards used in DRL algorithm. In section IV, the training and test setup is explained and the performance analysis of the training and test results is accessed. Finally, in section V, conclusions and the future work is presented.

II. Problem Statement

Reinforcement learning (RL) is an approach to artificial intelligence inspired in humans' way of learning, based on trail and error experiences. In RL agents are the computerized systems that learn, and the trial and error experiences are

obtained by interacting with the environment. Using the information about the environment state, the agent makes decisions and takes actions in discrete intervals known as steps. Each action updates the environment and its state. It produces also a reward, this is, a scalar value, submitted by the environment, informing about the benefit or inconvenient of such action. In this paper, the drone,a quadcopter, is used as an agent (the learner) and the agent will be rewarded in each time step by the environment. The actions executed by the agent are the inputs to the environment and the states and rewards will be the outputs of the environment. The objective of the agent is to maximize the final reward by learning which is the best action for each state.

Deep RL proposes the use of deep neural networks as the agent's decision algorithm. In conjunction with the experience replay memory, deep RL has been able to achieve super-human level when playing video and board games [4]. The deep RL solution is based in double deep Q-network (DDQN) [5], an extension of the DQN implementation [6]. DDQN selects from the state the action of the agent which maximizes the Q-value. Q-values are estimations of the future reward of an action executed in a given state.

In contrast with previous work [7], where relevant scalar state information was embedded into the image, this paper uses directly the scalars as part of the state, together with the image state data. To process such state a neural network that joints the two state parts into a unique flow is designed. It is named as Joint Neural Network (JNN). The image is the input of a convolutional neural network (CNN) and then a concatenation layer joints the flatten output of the CNN with the scalar values of the state.

A. Deep Reinforcement Learning

Reinforcement Learning is about learning from interaction how to behave in order to achieve a goal. The learner and decision-maker is called *agent*, while the thing it interacts with and therefore everything outside of the agent, is called the *environment*. The interaction takes place at each of a sequence of discrete time steps t. At each time step, the agent receives a *state* S_t from the *state space* S and selects an *action* A_t from the set of possible actions in the *action space* $A(S_t)$. One time step later the agent gets a numerical *reward* $R_{t+1} \subset \mathbb{R}$ from the environment as consequence of the previous action. Now the agent finds itself in a new state S_{t+1}.

The specification of their interface defines a particular task: the actions are the choices made by the agent; the states are the basis for making the choices, and the rewards are the basis for evaluating the choices. The Figure 1 illustrates this agent-environment interaction.

At each time step t, the agent maps from states to probabilities for selecting each possible action. This mapping is called the agent's *policy* π_t, with $\pi_t(a|s)$ as probability that $A_t = a$ if $S_t = s$.

$$\pi_t(a|s) \doteq P(A_t = a|S_t = s) \tag{1}$$

All reinforcement learning methods specify in their way how the policy is changed as a result of the agent's experience.

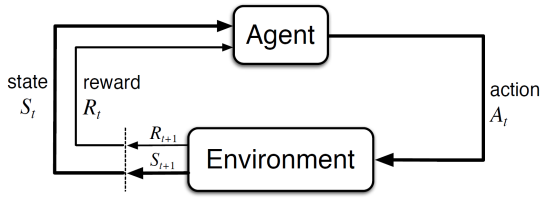

Fig. 1. The agent-environment interaction in reinforcement learning

Informally, the goal is to choose a policy so that it maximizes the total amount of reward. This means maximizing not the immediate rewards R_{t+1}, but the cumulative reward over time called return G_t.

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{2}$$

where $\gamma \in [0,1]$ is the *discount factor*. The discount factor γ determines the importance of future rewards. A factor of 0 will make the agent short-sighted by only considering current rewards, while a factor approaching 1 will make it strive for a long-term high reward. Surveys of reinforcement learning and optimal control [8], [9] have a good introduction to the basic concepts behind reinforcement learning used in robotics.

1) Deep Q-Network: The overall goal of Deep Q-Network (DQN) is to use a deep convolutional neural network to approximate the optimal action-value function, defined as:

$$\theta_\pi(s,a) = \max_\pi \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \mid s_t = s, a_t = a, \pi] \tag{3}$$

The optimal action-value function represents the maximum of the sum of rewards r_t discounted by γ at each time-step t, achievable by a behaviour policy μ = P(a|s), after making an observation (s) and taking an action (a).

The release of the DQN (Deep Q-Network) paper by DeepMind [4] noticeably changed Q-learning introducing a novel variant with two key ideas.

The first idea was using an iterative update that adjusted the action-values (Q) towards target values ($\gamma \max_a$ Q(s_{t+1},a)) that were only periodically updated, thereby reducing correlations with the target.

The second one was using a biologically inspired mechanism named experience replay that randomizes the data removing correlations in the observations of states and enhancing data distribution, with a higher-level demonstration and explanation by previous research in [10] , [11] and [12]. The use of the experience replay encourages the choice of an off-policy type of learning, such as Q-learning, because if not, past experiences would have been obtained following a different policy from the current one.

The whole process consists in characterizing an approximate value function $Q(s,a;\theta_i)$ using the CNN shown in eqn. 5, in which θ_i are the weights of the Q-network at iteration i. For the experience replay, agents experiences e_t which consist in

the tuple $(s_t, a_t, r_{t+1}, s_{t+1})$ are stored at each time-step t in the replay memory $e_1, \cdots, e_N$, where N sets the limit of entries, with the possibility of replacing older experiences for new ones when the limit of the memory is reached.

The standard Q-learning update for network parameters θ after taking action A_t in state S_t and observing the immediate reward R_{t+1} and resulting state S_{t+1} is:

$$\theta = \theta_t + \alpha[y_t^Q - Q(S_t, A_t; \theta_t)]\nabla_{\theta_t} Q(S_t, A_t; \theta_t), \quad (4)$$

where the estimated return as defined as Q-target y_t^Q:

$$y_t^Q = R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta) \quad (5)$$

This update resembles stochastic gradient descent, updating the current value $Q(S_t, A_t; \theta_t)$ over the Temporal Difference (TD) [13] error towards a target value y_t^Q.

2) Double Deep Q-Network: The idea proposed in [14] and named as Double Q-learning is basically based in decoupling action selection from evaluation.

As only one estimate is updated per step in a random selection, but two estimates are learned, it doubles the memory requirements but not the computational effort made at each step. The Double Q-learning was extended for the DQN-algorithm in [5]. The Double DQN algorithm remains the same as the original DQN-algorithm, except replacing the target the estimated return as defined as DQN-target y_{DQN} explained in [13].

III. Tools & Methods

A. Tools

1) AirSim Simulator & Unreal Engine: AirSim is a platform for AI research to experiment with deep learning, computer vision and reinforcement learning algorithms for autonomous vehicles [15], and it is built on Unreal Engine [16]. Unreal Engine provides ultra realistic rendering and strong graphic features for the Airsim. Quadcopter used in the Airsim simulator can be seen in Fig. 2.

Airsim has a lot of environments available to be used in reinforcement learning. These environments are mountains, blocks, city environment etc.

2) OpenAI-Gym: Gym [17] is an open source interface to reinforcement learning tasks. It is a toolkit for developing and comparing reinforcement learning algorithms. It is compatible with any numerical computation library, such as TensorFlow [18] or Theano [19]. The gym library has a collection of environments to test reinforcement learning algorithms. These environments have a shared interface which allows to write general algorithms.

3) Keras-rl: Keras-rl [20] implements some state-of-the art deep reinforcement learning algorithms in Python and seamlessly integrates with the deep learning library Keras. Keras can work with OpenAI Gym and it is built according to the developer needs, giving the ability to define own callbacks and metrics.

The model is developed in Keras, hold a replay buffer (experienced replay) of 50.000 transitions and follow a linear annealed policy from a maximum value 1.0 to a minimum value 0.1 until the final exploration step [1]. The Adam optimization [21] is used for the agent optimization to improve performance of training and testing of neural networks.

Fig. 2. Front-view of the quadcopter

B. Deep Reinforcement Learning Model

In previous work [7] the authors built a deep RL solution for a drone flying in an artificial environment built using the Unreal Engine blocks [16]. Three DQN algorithms were tested: DQN, double DQN (DDQN) and Dueling. The best results were obtained by DDQN, which showed to be at the level of a human tester. In this paper DDQN is chosen as the best algorithm, and a number of improvements will be added considering a new environment with moving obstacles, geo-fencing concept, and new scalar values which are added to the joint and the smoothing of the drone movements.

1) The Environment: Airsim is used for artificial intelligence research on deep learning and reinforcement learning algorithms for autonomous vehicles and it provides many environments which are developed as an Unreal plugin that can simply be dropped into any Unreal environment [22]. In this paper Airsim release v1.2.3 is used and small urban neighbourhood is selected as an environment. The neighbourhood environment is shown in Fig. 3.

Airsim also provides APIs to retrieve data and control vehicles autonomously. In Table I the data received from the Airsim is shown. The data contains the drone parameters in x and y directions such as the velocities of the drone (v_x, v_y), the distance between the goal and the drone in two directions (d_x, d_y) and the total distance to the goal (d_t) in the environment, the distance to the geo-fence limits (dg_{xmin}, dg_{xmax}, dg_{ymin}, dg_{ymax}). The geo-fence limits are shown in shaded blue region in Fig. 3. Also, drone yaw angle (ψ) relative to the initial orientation is received. DepthImage and the boolean landing and collision information are collected.

Fig. 3. The neighbourhood environment

TABLE I
DATA RECEIVED FROM THE ENVIRONMENT AT EVERY TIME-STEP

Data	Meaning
$v_x v_y$	agent's velocities in x and y directions
$d_x d_y d_t$	agent's distances to goal in x and y and total
dg_{xmin} dg_{xmax}	
dg_{ymin} dg_{ymax}	agent's distances to geo-fence limits
ψ	yaw angle relative to initial orientation
DepthImage	depth image in camera plan (256 x 144)
arrived	boolean landing info
collided	boolean collision info

A geo-fence is defined as a technology which creates a virtual barrier around a georaphical area. The geo-fence technology is used in drone navigation in order to create constraints for drones. The purpose of using the geo-fencing is to keep the drones out of or within the predefined area [23]. If the geo-fenced area is violated, the user can be notified to a pre-programmed entity and send warning signals to the operator to prevent the device entering this geo-fenced area. In a study [24] about a generic and modular geo-fencing strategy for civilian UAVs, geo-fence is used to avoid collisions with the environment such as controlled airspace areas, people, or other flying vehicles.

2) States: The states containing image and several scalars are processed by neural network that joints the two state parts into a unique flow called as Joint Neural Network (JNN). The image is the input of a convolutional neural network (CNN) and then a concatenation layer joints the flatten output of the CNN with the scalar values of the state. The part of the state containing the front depth image is set as 20x100 pixels image and the scalar values of the state are: the current velocity of the drone, the distance to the goal, angle to goal (track angle), elevation angle between the goal and the drone and the distance to the geo-fence limits.

3) Actions: In this paper, the UAV agent is flying at a fixed altitude. It is aimed that the drone collision can be observed by moving only along two dimensions (x and y). The output of the JNN architecture contains 5 different options: 4 actions which are the modification of the velocity in the two dimensions, the equivalent of $\pm 0.5 m/s$, and no speed modification option. Fig. 4 shows the 2D representation of the actions.

4) Rewards: The reward function is shown in Table II. The agent is rewarded $+100$ if the episode is successful by reaching the destination. On the other hand, the agent is penalized and it is given -100 if there is a collision or a violation of the geo-fence since the agent is the responsible for this collision. In this paper, the geo-fence is used as a virtual barrier for the drone and the drone is penalized if the predefined area is violated in order to increase the safe integration of drones into the environment. This is accomplished by calculating the distance to the geo-fence limits. The agent is also penalized in case of a collision if the agent is innocent and it is given -10. We consider the learner drone to be innocent when it can not see the random drone and it is the random drone movement which produces the collision. Independent of its source, a collision of the learner drone always ends the episode. Intermediate steps return a reward of -1 (to penalize delays) plus Δdg which is distance-to-the-goal difference with respect to the previous step. Δdg is used to stimulate actions that approach to the goal.

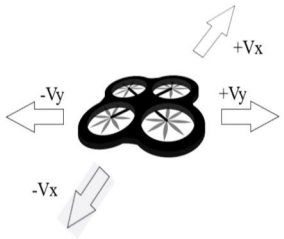

Fig. 4. Action Space

TABLE II
REWARDS

Reward	The Reason
$+ 100$	Goal reached
$- 100$	Collision: Obstacle (stationary or moving) or geo-fence
$- 10$	Collision: By random drone
$-1 + \Delta dg$	Otherwise

IV. PERFORMANCE ANALYSIS & RESULTS

A. Training and Test Setup

The training and testing of the drone is processed on a desktop with NVIDIA GeForce GTX 1060 with 6GB RAM graphic co-processor and Intel i7 processor, 16GB of memory.

B. Training Results

In this study, training divided into three sections: training number 1 including only learner drone, training number 2 including learner drone and random drone 1, and finally training number 3 having the learner drone and random drones 1 & 2. In each training, the learner drone is started with different yaw angles in order to increase the exploration capabilities of the learning.

In Fig. 5 the stationary positions of the random drones and the learner drone in the environment are shown. The random drones are used as moving obstacles and each random drone moves randomly but restricted to stay along the road.

For example, the area shaded with blue, and cyan colors represent the regions of random drones 1 & 2 respectively. Their duty is to block the movement of the drone trained by deep reinforcement learning.

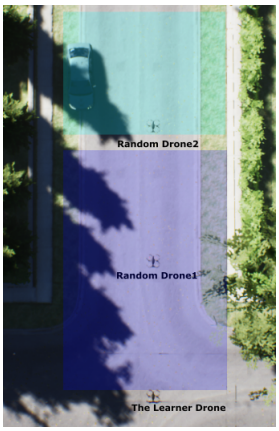

Fig. 5. The environment with Random Drones and the Learner Drone

In this section the training results of the 3 training sessions are analyzed. In Fig. 6, Fig. 7 and Fig. 8, the training results with 125000 steps and accumulated rewards are shown. The light blue represents the actual reward value of the step and the dark blue represents the mean rewards of the every 100 steps. The vertical dotted line represents the end of the annealing training part which is point at the 50000 steps. The training episodes are finished when the drone reaches the destination or when the learning drone collides with any stationary or moving obstacle (random drone) or when the learning drone overpasses the geo-fence limits.

The training results obtained by using only the learner drone and without any moving obstacles in the environment are shown in Fig. 6. It is seen that at the beginning of the training the rewards are below the zero when the random behavior is high. Although there are some successful episodes, the reward values are lower than the -100 which means the drone crashed. As the random behavior decreases over time, it is seen that the reward values are starting to increase and the trend of training curve is moving up and having more positive rewards. After annealing the reward values becomes more stable towards the end of the training.

In Fig. 7 the training results of the learner drone with "random drone 1" are shown. The training curve shows similar trend as in Fig. 6. For example, after around 25000 time steps, the learner drone starts learning how to avoid obstacles and the mean reward line goes up, the accumulative reward (R) increases. However, there exist episodes which crashed mostly at the beginning of the training and the reward values are between 0 and (-25). There are 135 episodes which the agent is hit by the "random drone 1". This random drone is the reason of the collision and the learning drone is innocent and then the learner drone penalized as (-10) and the simulation is reset. These crashes are represented in red cross. The crashes are mostly happened in the beginning of the training where the

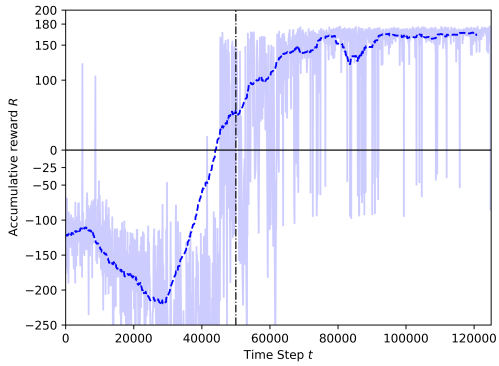

Fig. 6. Training results with only random drone 1

random behavior is high and also before the annealing point. As the training proceeds, the number of crashes caused by a randomly moving obstacle decreases and after some point there are no crashed episodes because of the random drone.

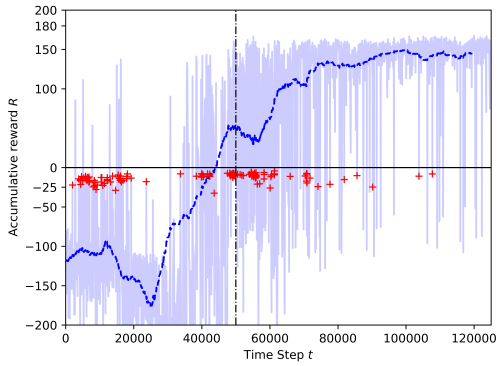

Fig. 7. Training results with random drone 1

In Fig. 8 the training results of the learner drone with 2 random drones are shown. The training curve shows similar trend as in Fig. 7. However, there exist 79 episodes which crashed because of the random drones. The learner drone is hit by the random drones in these crashes. Most of the crashes are happened in the beginning of the training where the random behavior is high. For example, almost half of the crashes happened before 20000 steps. After this point the learner drone starts exploring different parts of the environment where the random drones do not exist and thus there are no random crashed episodes until around 40000 steps. After that, the learner drone is hit by the random drones couple of times again but less than the beginning of the training. After a short time, the learner drone is starting to learn again to avoid obstacles and the mean reward values increase. The crashes caused by the random drones decrease through the end of the training and after some point the learner drone does not collide with anything.

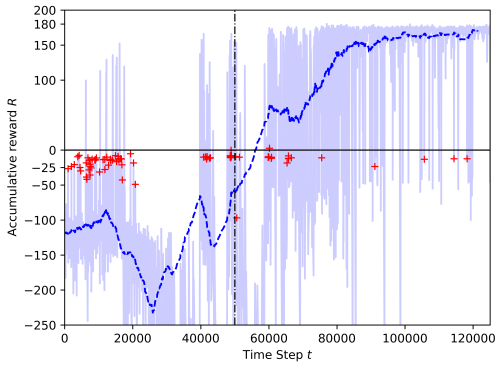

Fig. 8. Training results with random drones 1 & 2

C. Test Results

In this part, tests results of each training are shown. The tests are performed by using models created in each training section to observe how the agent can learn to avoid obstacles and the performance of the agent is assessed. These tests are made of 100 episodes starting from the take-off and ending by landing to the destination, by collision or by the time limit.

Test results are shown in Fig. 9 and Fig. 10. In Fig. 9 two scenarios are shown. First scenario has only learner drone in the environment and the second scenario includes the learner drone with "random drone 1" in the environment. In these tests, the learner drone has reached the destination without crashing anything. As it is seen in these figures, the cumulative reward values follow almost a straight line and having few oscillations. The details about the reward values of the tests and the number of successful episodes can be seen in Table III. The mean reward value for the model including the learner drone and the "random drone 1" is 163.49, lower than the mean reward for only one drone,learner drone, model, 172.63. This is because the learner drone has to deal with the moving obstacle in the second training session and the model has a reward lower than the reward in the first training model.

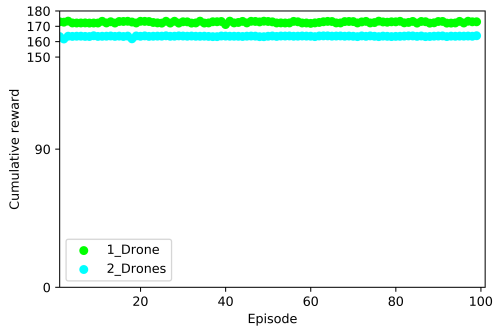

Fig. 9. Test results for only 1 drone and 2 drones

In Fig. 10 the test consists of 3 drones, the learner drone trained in the environment and random drones 1 & 2. In Fig. 10, green dots represent the successful episodes, red dot represents the crash and yellow dots which represent the timeout limit meaning there is no crash or successful episode, but only reached a timeout. There are 98 successful episodes out of 100 episodes and only one crash and one timeout happened in the test. The reward values can be seen in Table III. The mean reward value which is 153.69 is the lowest one when we compare with the other two tests. The reason can be more than one random drones exist in environment and the learning drone try to avoid them. Surprisingly, the maximum reward is the highest in all three tests.

The random behavior of the random drones which exist in the environment can change the path followed by the learner drone. For this reason, There are also successful episodes having reward values range between 90 and 150. The learner drone tries to avoid the random drones and the other stationary obstacles in the environment. Even if the reward values are lower, the learner drone successfully reaches the destination.

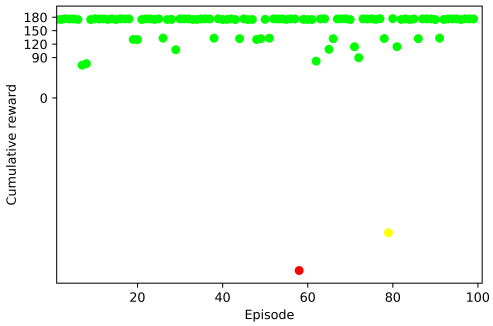

Fig. 10. Test results for 3 drones

TABLE III
MODEL TESTS COMPARISON

Model	Mean reward	Max reward	Min reward	Success rate
1 Drone	172.63	173.35	171.13	%100
2 Drones	163.49	164.02	161.72	%100
3 Drones	153.69	176.76	-385.21	%98

V. CONCLUSIONS & FUTURE WORK

In this paper, it is presented that a drone, quadrotor, can avoid both stationary and moving obstacles and navigate through the environment, small urban neighbourhood, by using deep reinforcement learning. The state containing image and several scalars are processed by neural network that joints the two state parts in order to increase the efficiency of the neural network.

The main contribution of this study has been introducing moving obstacles such as random drones in the environment and training of the drone to avoid both stationary and moving obstacles by using the deep reinforcement learning algorithm with joint neural network.

Artificial intelligence subject has always been improving itself and new ideas emerge. In the future, multi-agent deep

reinforcement learning and transfer learning subjects are considered to be studied and implemented to DRL algorithm proposed in this paper. It is believed that multi-agent DRL methods can improve the training and test results by using intelligent drones which communicate with each other via sharing information in the environment. By transfer learning, the aim is to increase the performance of the training time spent on training by using a learning model constructed before. In addition, the linear annealed policy can be replaced by logarithmic annealed policy because we believe that the time spent on random behavior is too high in training by using linear annealed policy. Finally, the hardware can be upgraded to improve the training.

ACKNOWLEDGMENT

This work has been funded by the Ministry of Science, Innovation and Universities of Spain under grant number TRA2016-77012-R

REFERENCES

[1] G. Muñoz Ferran, "A new deep reinforcement learning architecture for autonomous uavs," B.S. thesis, Universitat Politècnica de Catalunya, 2018.

[2] J. Duo and L. Zhao, "Uav autonomous navigation system for gnss invalidation," in *2017 36th Chinese Control Conference (CCC)*, pp. 5777–5782, IEEE, 2017.

[3] H. X. Pham, H. M. La, D. Feil-Seifer, and L. V. Nguyen, "Autonomous UAV navigation using reinforcement learning," *CoRR*, vol. abs/1801.05086, 2018.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[5] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning.," in *AAAI*, vol. 2, p. 5, Phoenix, AZ, 2016.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013.

[7] K. Kersandt, G. Muñoz, and C. Barrado, "Self-training by reinforcement learning for full-autonomous drones of the future," in *Digital Avionics Systems Conference (DASC), 2018 IEEE/AIAA 37th*, pp. 1–10, IEEE, 2018.

[8] B. Kiumarsi, K. G. Vamvoudakis, H. Modares, and F. L. Lewis, "Optimal and autonomous control using reinforcement learning: A survey," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 6, pp. 2042–2062, 2018.

[9] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

[10] M. B. L. . O. R. C. McClelland, J. L., "Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionist models of learning and memory," in *Psychological Review, 102(3)*, pp. 419–457, 1995.

[11] M. Riedmiller, "Neural fitted q iteration-first experiences with a data efficient neural reinforcement learning method," in *ECML*, vol. 3720, pp. 317–328, Springer, 2005.

[12] L.-J. Lin, *Reinforcement Learning for Robots Using Neural Networks*. PhD thesis, Pittsburgh, PA, USA, 1992. UMI Order No. GAX93-22750.

[13] K. Kersandt, "Deep reinforcement learning as control method for autonomous uavs," Master's thesis, Universitat Politècnica de Catalunya, 2018.

[14] H. V. Hasselt, "Double q-learning," in *Advances in Neural Information Processing Systems*, pp. 2613–2621, 2010.

[15] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*, 2017.

[16] "Unreal engine 4." https://www.unrealengine.com/en-US/what-is-unreal-engine-4. Last Accessed: 2019-01-29.

[17] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[18] M. A. et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[19] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016.

[20] M. Plappert, "keras-rl." https://github.com/keras-rl/keras-rl, 2016.

[21] D. P. Kingma and J. L. Ba, "Adam: Amethod for stochastic optimization," in *Proc. 3rd Int. Conf. Learn. Representations*, 2014.

[22] "Airsim documentation." https://microsoft.github.io/AirSim. Last Accessed: 2019-05-01.

[23] F. von Bothmer, *Missing Man: Contextualising Legal Reviews for Autonomous Weapon Systems*. PhD thesis, Universität St. Gallen, 2018.

[24] T. Gurriet and L. Ciarletta, "Towards a generic and modular geofencing strategy for civilian uavs," in *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 540–549, IEEE, 2016.